Olay Tabanlı Bir Yazılım Mimarisinde Bağımlılık İletimi ve Bileşen Gerçekleştirimi

Orçun Dayıbaş[1], Serdar Doğan[2]

Aselsan A.Ş. SST-MD-YMM, P.K. 1 06172, Yenimahalle, Ankara
odayibas@aselsan.com.tr1
serdardogan@aselsan.com.tr2

Özet. Olay yolu (Event Bus), yazılım bileşenleri arasında Yayımcı-Abone (Publish-Subscribe) tarzına uygun iletişim sağlayan ve bunu yaparken yazılım bileşenlerinin birbirlerinden haberdar olmasını gerektirmeyen bir yapıdır. Bağımlılık iletimi (Dependency Injection) de bileşen bağımlılıklarının dışarıdan iletilerek bileşenlerin yapılandırılabilmesine olanak sağlar. Mimarisi olay yolu üzerine kurgulanmış bir yazılımda bağımlılık iletimi kullanılması durumunda, bu iki yaklaşımın etkileşimi, denetim altına alınması gereken bir husus haline gelir. Bu çalışma kapsamında, sözü edilen denetimin yazılım geliştirme süreci içerisinde nasıl ele alınabileceği tartışılmıştır. Bu bağlamda, geliştirdiğimiz derinlik ölçüm sonarı (iskandil sistemi) grafiksel kullanıcı arayüz (GKA) yazılım mimarisine temel olan "Yolcu" çerçevesi ve ilgili proje kapsamında edinilen kullanım deneyimleri de makale kapsamında verilmiştir.

Anahtar Kelimeler. Bağımlılık iletimi, Olay tabanlı yazılım mimarisi, Yeniden Kullanım, Olay yolu.

1 Giriş

Yazılım dünyasında üretkenliğin artırılması için ortaya konulan en temel yöntemlerden biri yeniden kullanımdır. Yeniden kullanımın sistematik olarak desteklenmesi için yazılım bileşenlerinin tasarım ve geliştirim süreçlerinin, bu temel üzerine kurulması gerekir. Bu bağlamda yazılım mimarisi, yeniden kullanımı garanti altına almak (sistematik hale getirmek) için bir araç olarak görülebilir. Yeniden kullanım odaklı bir mimari tasarımda, nesne yönelimli tasarımın temel ilkelerinin (SOLID) gözetilmesi gerektiği söylenebilir [1]. Bu temel ilkelerin hepsi ya da bir kısmının, mimari tarafından bileşen seviyesinde zorlanması yazılım kalitesini ve üretkenliğini artıracak bir unsurdur.

Bu makale kapsamında ele alınan durum çalışması, Aselsan bünyesinde tanımlı, derinlik ölçüm sonarı grafiksel kullanıcı arayüzü yazılımı geliştirme sürecini kapsamaktadır. İlgili yazılım, belirli bir proje ailesi (KULAÇ) için ortak kullanılması öngörülerek geliştirilmiştir. Yazılım geliştirme ve analiz sürecinde yetenek tabanlı bileşenler (ve yetenek gereksinimleri) çıkartılarak, yeni gelecek benzer projelerde de bu varlıkların yeniden kullanılması hedeflenmiştir. Burada kurulması (o tarihte) öngörülen altyapı, kısmen [2]'de tanımlanmış, bu makale kapsamında da tanımlanan

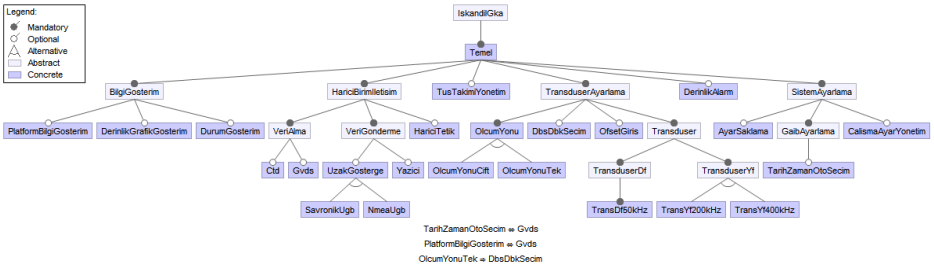
yöntemin nasıl uygulandığı ele alınmıştır (Yetenek modellerinin nasıl ele alındığına dair detaylı bilgi için bkz. [2]).

Makalenin geri kalanı şu şekilde düzenlenmiştir: İkinci bölümde yeteneklerin varlıklarla nasıl ilişkilendirildiği ve yazılım gereksinim belgesi üzerinde nasıl ifade edildiği verilmiştir (Problem alanının çözümlenmesi). Üçüncü bölüm, bileşenler arası etkileşimlerin mimari seviyede nasıl ele alınacağı tartışılmış ve hemen ardından dördüncü bölümde bu çalışmanın sonucu olarak ortaya konulan yeniden kullanılabilir çerçevenin (Yolcu) detayları ve bileşenler arası etkileşimlerin bu çerçevede nasıl ele alındığı belirtilmiştir. Çalışmanın sonuç ve değerlendirmesi de beşinci bölümde verilmiştir.

2 Yetenek Modeli ve Gereksinim Yönetimi

Yazılım bileşenlerinin etkin yeniden kullanımı için gereksinim yönetimi önemli bir unsurdur [3]. Değişkenlik yönetiminin yetenek tabanlı [4] yapıldığı bir ortamda, gereksinimlerin tanımlarının da yetenekler ile ilişkilendirilmesi ihtiyacı ortaya çıkmaktadır. Bu amaçla, projelerde kullandığımız YGÖ - Yazılım Gereksinim Özellikleri (SRS) şablonuna aşağıdaki eklemeler yapılmıştır[1]:

- *Yetenek Modeli:* Yetenek modeli, yetenekler arasında sıradüzensel ve karşılıklı ilişkileri gösteren modeldir [4]. Proje kapsamında, yetenek modeli ağaç şeklinde görselleştirilerek gereksinim belgesine (YGÖ) eklenmiş (bkz. Şekil 1) ve geçerli yazılım ürünlerinin, modelin geçerli yapılandırmaları olacağına dair bir tanım maddesi eklenmiştir.



Şekil 1. KULAÇ GKA Yetenek Modeli

- *Yetenek Bilgisi (Yetenek Kolonu):* Gereksinimlerin ilgili olduğu yetenek(ler), gereksinim yönetim aracı (Doors) üzerinde tanımlı şablona bir özellik olarak eklenmiş ve yeni bir kolonda verilmiştir (bkz. Şekil 2). Bu sayede gereksinimler yetenekler üzerinden sorgulanabilir hale gelmiştir (Doors'un sorgulama altyapısı kullanılarak).

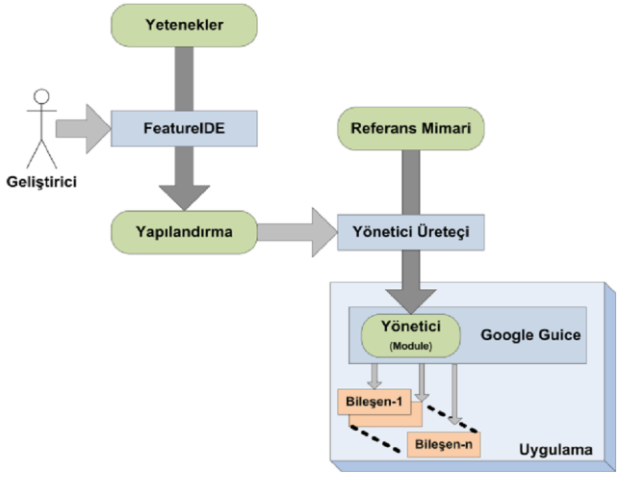
[1] Sürecin temel aldığı şablon J-STD-016 standardına dayanmaktadır [5].

ID		Nitelik	Yetenek
KAYKB_YGÖ.236	**3.2.6.9 *Derinlik Alarmı İşlemleri***	Başlık	
KAYKB_YGÖ.237	[redacted]	Gereksinim	DerinlikAlarm
KAYKB_YGÖ.603	[redacted]	Gereksinim	DerinlikAlarm OlcumYonuCift
KAYKB_YGÖ.238	[redacted]	Tanım	
KAYKB_YGÖ.239	[redacted]	Gereksinim	DerinlikAlarm
KAYKB_YGÖ.240	[redacted]	Gereksinim	DerinlikAlarm TransDf50kHz
KAYKB_YGÖ.241	[redacted]	Gereksinim	DerinlikAlarm TransDf50kHz
KAYKB_YGÖ.242	[redacted]	Gereksinim	DerinlikAlarm
KAYKB_YGÖ.243	[redacted]	Gereksinim	DerinlikAlarm
KAYKB_YGÖ.244	[redacted]	Gereksinim	DerinlikAlarm
KAYKB_YGÖ.245	[redacted]	Gereksinim	DerinlikAlarm
KAYKB_YGÖ.246	[redacted]	Gereksinim	DerinlikAlarm
KAYKB_YGÖ.248	[redacted]	Gereksinim	DerinlikAlarm AyarSaklama

Şekil 2. Doors Şablonuna Eklenen Yetenek Kolonu[2]

Yetenek gereksinimlerinin açıkça ifade edilmesi, hem bu yetenekleri sağlayan bileşenlerin gereksinimlerine kolayca erişebilmek, hem de bileşen bağımlılıklarını çıkarabilmek adına yarar sağlamıştır. Yetenek modelinin bir diğer faydası da sıradüzensel bağımlılıkların, gereksinim gerçekleştirim önceliklerini belirlemek için kullanılabilmesidir. Örneğin yetenek ağacındaki her bir yaprak düğümün (leaf node) gerçekleştirim önceliği kendisini köke (temel yetenek) bağlayan düğümlerin gerçekleştirim önceliğine eşit veya düşük olmalıdır. Aynı yorum, doğrudan gerektirme ("requires" türü) bağımlılıkları için de yapılabilir.

Yetenek modellerinin gerçekleştirimine ilişkin önerdiğimiz yaklaşım, [2]'de detaylı olarak verildiğinden, bu makalenin kapsamı dışında tutulmuştur. Sözü edilen yaklaşıma dair genel akış, Şekil 3'de gösterilmiştir.



Şekil 3. Yetenekten Gerçekleştirime Yarı Otomatik Geçiş [2]

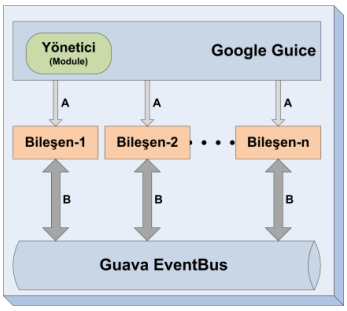
[2] Madde metinleri ticari gizlilik açısından karartılmıştır.

3 Yazılım Mimarilerinde Bileşen Etkileşimleri

Nesne yönelimli bir mimaride, yazılım bileşenleri arasındaki etkileşimler iki temel sınıfta ele alınabilir; olgu yaratma ve iş akışı çağrıları:

- *Olgu Yaratma:* Bir nesne, bağımlı olduğu soyutlama (Ör: Arayüz tanımı) üzerinden bir nesnenin olgusunu yaratamaz. Ayrıca nesnelerin olgularının yaratılması soyutlama üzerinden yapılsa da nesneye doğrudan bağımlılık oluşturur. Bu durum, "*Soyutlamalara bağımlı ol, gerçekleştirimlere bağımlı olma*" ilkesinin ihlali anlamına gelir [1]. Bu durumu ortadan kaldırmak için soyut fabrikalar [6], bağımlılık iletimi [7] vb. yöntemler yaygın olarak kullanılmaktadır. Bu çalışma kapsamında nesne yaratımına ilişkin detayların uygulama kodundan tamamen soyutlanması ve sınanmış bir çerçeve kullanımına olanak tanımak adına açık kaynaklı bir bağımlılık iletimi çerçevesi olan "Google Guice" [8] kullanımı tercih edilmiştir (bkz. Şekil 3).
- *İş Akışı Çağrıları:* Nesneler bir araya gelerek bir kullanım senaryosunu işletirken, çoğunlukla belirli bir iş akışını izleyen çağrılar yaparlar. Bu çağrıların doğrudan yapılması bağlaşımı (coupling) artıran ve dolayısı ile yeniden kullanımı zorlaştıran bir unsurdur. Bu duruma çözüm olarak da servis tabanlı [9], olay tabanlı [10] vb. mimarilerin kullanımı değerlendirilebilir. Başarımı en az etkileyecek şekilde bağlaşımı azaltacak bir çözüm bulmak ve bir önceki maddede belirtildiği gibi yine sınanmış bir çerçeve kullanmak adına "Google Guava EventBus" [11] kullanımı tercih edilmiştir.

Olay tabanlı bir mimaride çağrıyı yapan ve çağrıya cevap veren bileşenler arasında doğrudan bir bağımlılık yoktur. İş akışı, "Olay Yolu" üzerinden iletilen "olay"lar ile gerçekleşir (bkz. Şekil 4). Olay yolu, yayınlanan bir olayı, o olayı dinleyen tüm abonelere iletir (Yayımcı-Abone). Bu yaklaşımda, her ne kadar yazılım bileşenlerinin birbirlerine doğrudan bağımlılıkları olmasa da iletişim için ilgili olay türlerini bilmeleri gerekir (Bileşenler olay türlerine bağımlıdır). Bu bağlamda, birbirlerinin arayüzlerini bilmeyen bileşenler arasındaki iletişimi sağlayabilmek için ortak bir dilin tanımlanması ihtiyacı doğmaktadır. Bu dil, olay yolu üzerinden gidip gelecek nesne türlerinin (olay) belirlenmesi ile tanımlanır.



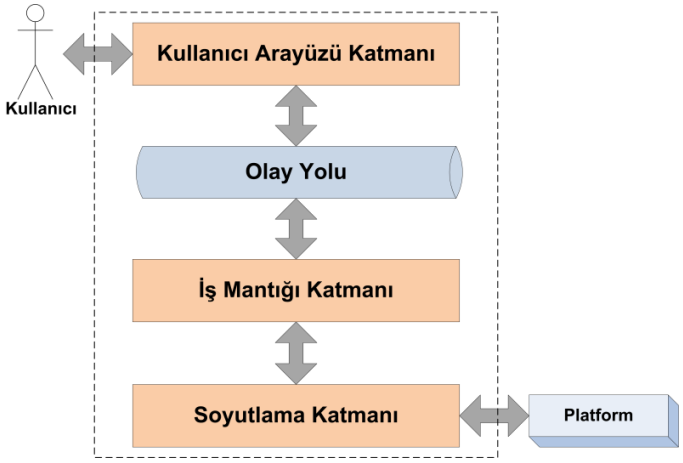
Şekil 4. KULAÇ GKA Yazılımında Olgu Yaratma (A) ve İş Akışı (B) Bileşen Etkileşimleri

Olay tabanlı olmayan bir yapıdaki her farklı yöntem çağrısı, olay tabanlı yapıda yeni bir olay türüne karşılık gelmektedir. Bu durum, her ne kadar küçük ölçekli yazılımlar için kabul edilebilir olsa da yazılım ölçeği büyüdükçe çok sayıda olay türü tanımlanması, bazı yan etkilere yol açabilir. Bunlara örnek olarak kodun idame edilebilirliğinin azalması ve hata ayıklamanın zorlaşması verilebilir. Bu problemlerin oluşmasını önlemek adına önerilen çerçevede olay kullanımı kurallarla sınırlandırılmıştır (bkz. 4. Bölüm).

4 Yolcu Çerçevesi

Yolcu çerçevesi, veri paylaşımı ve iş akışı tetikleme altyapısı olarak olay yolu kullanımını benimseyen yazılımlar için geliştirilmiş bir çerçevedir. Yolcu çerçevesi kullanılarak geliştirilmiş bir uygulamanın merkezinde üç temel unsur yer alır; "Olay Yolu", "Kullanıcı Arayüzü Yönetici" (KA Yönetici) ve "İş Yönetici". Olay yolu, platform ya da kullanıcı kaynaklı olayların ilgili birimlere (aboneler) iletilmesinden sorumludur. "Kullanıcı Arayüzü Yönetici", yazılımın kullanıcı ile olan etkileşiminden sorumlu bileşenler (veri girişi, veri görüntüleme), "İş Yönetici" ise bir veya birden çok kullanım durumunun (senaryo) gerçekleştirilmesinde rol oynayan, iş mantığının gerçeklendiği ya da alt katman soyutlamalarının kullanımı ve yönetiminden sorumlu bileşenlerdir.

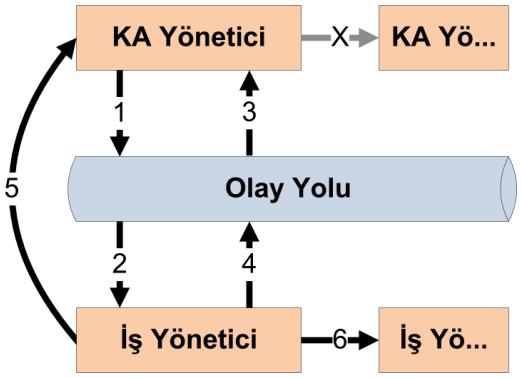
Yolcu çerçevesi temelde çok katmanlı bir grafiksel kullanıcı arayüzü uygulama geliştirme altyapısı sunar (bkz. Şekil 5). En üst seviyede kullanıcı arayüzü katmanı, altta iş mantığı katmanı ve aralarında olay tabanlı iletişimi sağlayan olay yolu bulunur.



Şekil 5. Katmanlı Mimari ve Yolcu Çerçevesinin Yeri

Olay yolu için olayı yayınlayan ya da dinleyen yöneticilerin hangi katmanda olduğunun ya da iletişimin hangi yönde olduğunun bir önemi yoktur. İki yönde de olaylar kaynağı ya da hedefi sorgulanmadan iletilir. İletişim yönü için bir kısıt olmaması, gerçekleştirim sırasında döngüsel bağımlılık kurulmasına neden olabilir. Bu

ve benzeri sorunların önüne geçmek için Yolcu çerçevesi, yazılım katmanları ve bileşenler arasındaki iletişimi sınırlandırmaktadır. İzin verilen bileşen etkileşimleri ve bu etkileşimlere dair açıklamalar aşağıda verilmiştir:



Şekil 6. Yolcu Çerçevesi Bileşen Etkileşim Kuralları

1. Kullanıcı Arayüz (KA) yöneticilerin yayınladığı olaylardır. Bir iş akışı başlatan kullanıcı girdileri (Ör: Kullanıcının bir menü düğmesine basması), olay olarak yayınlanır. Bir KA yönetici başka bir KA yöneticiye doğrudan erişmez ya da kullanıcı girdilerini ifade etmek için tanımlanmış olayları dinleyemez. KA yöneticilerin kullanım durumları kapsamında yönetim sorumluluğu, iş yöneticilere devredilerek, KA katmanındaki bileşen bağımlılıkları ortadan kaldırılmaya çalışılmıştır.
2. İş yöneticilerin dinlediği olaylardır. İş yöneticiler, kullanım durumu başlatan KA olaylarını ya da diğer iş yöneticilerin yayınladığı olayları dinlerler.
3. KA yöneticiler, iş yöneticilerin yayınladığı olayları dinlerler. Bunlar bir kullanım durumu kapsamında alt iş adımları olabileceği gibi alt katman soyutlamalarından sorumlu iş yöneticilerin yayınladığı olaylar (DDS verileri, seri kanal girdileri vb.) da olabilir.
4. İş yöneticilerin yayınladığı olaylardır. İş yöneticilerin yayınladıkları olaylar hem diğer iş yöneticileri hem de KA yöneticileri ilgilendiriyor olabilir.
5. Kullanım durumu alt adımları doğrudan yöntem çağrısı ile yapılır (zaman uyumsuz olarak yayınlanması gereken bir olay varsa yine yayınlanabilir). Bu bağımlılıklar, arayüzler üzerinden ve bağımlılık iletimi ile (bkz. 3. Bölüm) kurulduğundan, çalışma zamanında KA yönetici gerçekleştirimini değiştirmek de mümkündür.
6. Zaman uyumlu işletilen kullanım durumu adımlarında bir iş yönetici diğer bir iş yöneticiyi doğrudan kullanabilir. Bu bağımlılıklar da yine bağımlılık iletimi yöntemi ile kurulur.

5 Sonuç ve Değerlendirme

KULAÇ proje ailesi için GKA yazılımı geliştirme işleri kapsamında ortaya konulan ve bu makalede detayları verilen yaklaşımlar ile aşağıdaki kazançlar sağlanmıştır:

- Yeniden kullanılabilir varlıklar (bileşen ve gereksinimler), yetenek tanımları ile ilişkilendirilerek, üst seviye bir soyutlama ile projeler arası benzerliklerin ele alınabilmesi için ortam hazırlanmıştır. Toplam 316 gereksinim maddesi, 24 yetenek ile ilişkilendirilmiştir. Bu gereksinimlerin 199 adedi temel yetenek ile ilişkilendirilmiştir. Proje ailesinin büyümesi ile yetenek sayısının artacağı ve temel yeteneğin küçüleceği öngörülmektedir.
- Bileşenlerin olgu yaratma kodları bağımlılık iletimi çerçevesi, iş akışı çağrıları da olay yolu çerçevesi tarafından ele alındığından iş mantığı kodunda önemli bir sadeleşme elde edilmiştir. Ayar ve akış denetim kodlarının ("Boilerplate" kodlar) ortadan kalması, daha okunur ve daha kolay yönetilebilir bir kod yapısı oluşturulabilmesine olanak tanımıştır (bkz. Çizelge 1: KGBYKB, bir su üstü KULAÇ projesi için kodlanmış eski GKA yazılımı. KAYKB, yeni yaklaşımla kodlanmış, hem su üstü hem de su altı projesi için gerekli yeteneklerini kapsayan ortaklanmış GKA yazılımı).

Table 1. Çizelge **2.** KGBYKB (Su üstü) ve KAYKB (Denizaltı + Su üstü) Karşılaştırma Çizelgesi

Ölçüm[3]	KULAÇ KGBYKB (2011)	KULAÇ KAYKB (2013)
Abstractness	10.10%	16.10%
Average Lines Of Code Per Method	10.03	9.75
Efferent Couplings	285	200
Lines of Code	45,133	19,747
Number of Methods	4,057	1,458
Number of Packages	88	39
Number of Types	850	341
Weighted Methods	6,798	2,623

İlerleyen dönemde, sözü edilen kazançların başka proje ailelerinde de kullanılabilmesi adına geliştirilen Yolcu çerçevesi ve kullanım ilkelerinin diğer proje ekipleri ile de paylaşılması öngörülmektedir.

[3] Ölçümler için "CodePro Analytix" kullanılmıştır. Ölçüm isimleri, tekrarlanabilirliği sağlamak için araçta olduğu haliyle (İngilizce) verilmiştir. Daha ayrıntılı bilgi için: http://developers.google.com/java-dev-tools/code

Kaynaklar

1. R.C. Martin, “Design Principles and Design Patterns”, Object Mentor (2000).
2. O.Dayıbaş, “Yetenek Modellerinin Gerçekleştirimi Üzerine Bir Durum Çalışması”, UYMK (2012).
3. K.Pohl, “Requirements Engineering: Fundamentals, Principles, and Techniques”, Springer (2010).
4. K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, “Feature Oriented Domain Analysis (FODA) Feasibility Study”, Technical Report CMU/SEI-90-TR-021 (1990).
5. R. Sorensen, “MIL-STD-498, J-STD-016, and the U.S. Commercial Standard”, CrossTalk (1996).
6. E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison Wesley (1994).
7. M. Fowler, "Inversion of Control Containers and the Dependency Injection Pattern", http://martinfowler.com/articles/injection.html (2004).
8. “Google Guice – A Lightweight Dependency Injection Framework”, http://code.google.com/p/google-guice
9. N. Bartlett, “OSGi in Practice”, Draft Preview http://njbartlett.name/osgibook.html (2009).
10. D. Garlan “Formal Modeling and Analysis of Software Architecture Components, Connectors, and Events”, Formal Methods for Software Architecture LNCS Vol. 2804 (2003).
11. “Guava: Google Core Libraries for Java 1.6+”, http://code.google.com/p/ guava-libraries